Logic programs with abstract constraint atoms

Content areas: Knowledge representation, logic programming, nonmonotonic reasoning

Abstract

We propose and study extensions of logic programming with constraints represented as generalized atoms of the form $C(X)$, where X is a finite set of atoms and C is an abstract constraint (formally, a collection of sets of atoms). Atoms $C(X)$ are satisfied by an interpretation (set of atoms) M, if $M \cap X \in C$. We focus here on monotone constraints, that is, those collections C that are closed under the superset. They include, in particular, weight (or pseudo-boolean) constraints studied both by the logic programming and SAT communities. We show that key concepts of the theory of normal logic programs such as the one-step provability operator, the semantics of supported and stable models, as well as several of their properties including complexity results, can be lifted to such case.

Introduction

In this paper, we study logic programs whose clauses are built of generalized atoms expressing constraints on sets. We show that under the assumption of *monotonicity* of constraints, and with an appropriate handling of nondeterminism inherent in deriving ways to satisfy constraints, basic concepts, methods, semantics and results studied in normal logic programming generalize to the extended setting. Our work provides a theoretical framework for recent extensions of logic programming with weight constraints (Simons, Niemelä, & Soininen 2002) but applies to a much broader class of programs.

In the 1990s researchers demonstrated that normal logic programming with the stable-model semantics is an effective knowledge representation formalism. It provides solutions to problems arising in such contexts as planning, reasoning about action, diagnosis and abduction, product configuration, and modeling and reasoning about preferences. Moreover, due to the emergence of fast methods to compute stable models (Niemelä & Simons 1997; Leone *et al.* 2003; Babovich & Lifschitz 2002; Lin & Zhao 2002), the importance of the formalism increased significantly as it became possible to use it not only as a modeling language but also as a practical computational tool. (Baral 2003) and (Gelfond & Leone 2002) provide a detailed discussion of the formalism and its applications.

In the last few years, researchers proposed extensions of the language of normal logic programming with means to model constraints involving *aggregate* operations on sets. (Simons, Niemelä, & Soininen 2002) proposed a formalism integrating logic programming with *weight constraints*[1], generalized the concept of stable models to this extended setting, and developed fast algorithms to compute them. (Denecker, Pelov, & Bruynooghe 2001; Pelov, Denecker, & Bruynooghe 2004), introduced a formalism allowing for more general aggregates. They extended to this new setting several semantics of normal logic programs, including the stable-model semantics and the well-founded semantics. A related recent work (Dell'Armi *et al.* 2003), incorporated aggregates into the formalism of *disjunctive* logic programs with the *answer-set semantics*. Such extensions are important as they simplify the task of modeling problem specifications, typically result in more direct and more concise encodings, and often significantly improve the computational effectiveness of the formalism as a problem-solving tool[2].

However, arguably, all these approaches are subject to limitations. The formalism of (Simons, Niemelä, & Soininen 2002) admits only weight constraints and focuses almost entirely on the semantics of stable models. Furthermore, the connections to normal logic programming are indirect and somewhat obscure. There is a correspondence to logic programs with *nested* expressions (Ferraris & Lifschitz 2004), which helped establish some interesting results on programs with weight constraints (for instance, characterizations of their strong equivalence (Turner 2003)). However, the direct encodings of programs with weight constraints as programs with nested expressions have large size and, more importantly, it is not clear whether that connection applies in the broader context of other types of constraints. The other two approaches consider broader classes of constraints and are more firmly grounded in the core formalism of normal (dis-

[1]These are constraints on the total weight of elements in sets. In the SAT community, they are also referred to as *pseudo-boolean* constraints.

[2]For similar reasons, extending propositional logic with aggregates (most notably, weight constraints) has recently been gaining attention (Aloul *et al.* 2002; Dixon & Ginsberg 2002; Prestwich 2002).

junctive) programs, but are limited in that they do not allow aggregate constructs to appear in the heads of the clauses. In addition, all approaches discussed above focus on concrete constraints, and so, they are restricted in their applicability.

Our goal is to address these limitations. We develop a formalism of logic programs with clauses built of generalized atoms that express constraints on sets. To this end, we propose the notion of an *abstract constraint* and its linguistic counterpart — an *abstract constraint atom*. We then use abstract constraint atoms as building blocks of program clauses. We restrict our attention to *monotone* constraints, as monotonicity is essential for preserving a procedural reading of a logic program as a computational device. We show that basic concepts, techniques and results of normal logic programming have direct generalizations to the class of programs built of monotone constraints. In particular, we extend to that setting the *supported-model* and the *stable-model* semantics. We also establish some relevant complexity results. Our work is related to (Marek, Niemelä, & Truszczyński 2004). That work, however, focused only on one specific class of constraints — cardinality constraints.

The key tool in our work is the *nondeterministic one-step provability* operator. It generalizes the one-step provability operator of (van Emden & Kowalski 1976). Thanks to close parallels between these two concepts, we are able to reconstruct operator-based characterizations of models, supported models and stable models, presented in (Apt 1990; Fitting 2002). We also distinguish and discuss the class of *deterministic* programs (programs with clauses whose heads can be satisfied only in one way). For these programs the one-step provability operator becomes deterministic and the theory of normal logic programming extends to deterministic programs without *any* significant change.

Basic concepts, motivation, examples

We limit our considerations to the propositional case. As we interpret programs with variables by means of Herbrand models, the first-order case reduces to the propositional one.

We consider a language determined by a fixed countable set At of *propositional atoms*. An *abstract constraint* is a collection $C \subseteq \mathcal{P}(At)$. An *abstract constraint atom* is a syntactic expression of the form $C(X)$, where $X \subseteq At$ is *finite* and C is an abstract constraint.

We interpret abstract constraint atoms by means of propositional interpretations (truth assignments), represented as subsets of At[3]. An interpretation $M \subseteq At$ *satisfies* an abstract constraint atom $C(X)$ ($M \models C(X)$), if $M \cap X \in C$ (that is, if the set of atoms in X that are true in M belongs to, or satisfies, the constraint C). Otherwise, M does not satisfy $C(X)$, ($M \not\models C(X)$). In that case, we also say that M satisfies the *literal* $\mathbf{not}(C(X))$. An abstract constraint atom $C(X)$ is *consistent* if there is an interpretation M such that $M \models C(X)$. We will now illustrate these concepts with several examples of common constraints.

Weight constraints. Given a real number v and a weight function w, assigning to each atom in At a real number (its *weight*), a weight constraint $W^{\leq}_{w,v}$ imposes a restriction that "a total weight of atoms in an *allowed* subset of At be at least v". We represent $W^{\leq}_{w,v}$ as an abstract constraint $W^{\leq}_{w,v} = \{A \subseteq At \colon v \leq \sum_{a \in A} w(a)\}$ (comparison relations $<, >, \geq$ give rise to other types of weight constraints).

Volume constraints. They differ from weight constraints in that they restrict the product of individual weights of atoms in allowed sets, depending on the type of the comparison relation used. Selecting the relation $\leq$ and assuming the same notation as before, we express volume constraints as abstract constraints of the form $V^{\leq}_{w,v} = \{A \subseteq At \colon v \leq \Pi_{a \in A} w(a)\}$.

Maximum constraints. Given a weight function w on the set of atoms and a real bound v, the maximum constraint restricts allowed sets of atoms to those with the maximum weight at least v. Formally, we express them as abstract constraints of the form $Max^{\leq}_{w,v} = \{A \subseteq At \colon v \leq \max\{w(a) \colon a \in A\}\}$ (or it variants, depending on the comparison relation).

Even- and odd-cardinality constraints. They impose a parity requirement on the cardinality of allowed sets. Formally, we express them as abstract constraints $E = \{A \subseteq At \colon |A| \text{ is even}\}$ and $O = \{A \subseteq At \colon |A| \text{ is odd}\}$.

Containment constraints. Such constraints require that allowed sets contain some prespecified configurations (subsets). We capture them by abstract constraints $C_{\mathcal{A}}$ that consist of all subsets of At that contain at least one element from a prespecified collection $\mathcal{A}$ of finite subsets of At.

Each of these constraints determines associated abstract constraint atoms. Let $At = \{p_1, p_2, \ldots\}$ and let us consider a weight function w such that $w(p_i) = i$, $i = 1, 2, \ldots$. The expression $W^{\leq}_{w,6}(p_1, p_2, p_5, p_6)$ is an example of an abstract constraint atom. A set M of atoms satisfies $W^{\leq}_{w,6}(p_1, p_2, p_5, p_6)$ if and only if the total weight of atoms in $M \cap \{p_1, p_2, p_5, p_6\}$ is at least 6 (that is, whenever M contains p_6, or p_5 together with at least one other atom). Similarly, the abstract constraint atom $Max^{\leq}_{w,5}(p_2, p_4, p_6, p_8)$ enforces the restriction that allowed sets contain p_6 or p_8. An abstract constraint atom $E(p_1, p_7)$ (E stands for the even-cardinality constraint) forces allowed sets of atoms to contain none or both of p_1 and p_7. All these constraint atoms are consistent. An atom $W^{\leq}_{w,7}(p_1, p_2, p_3)$ is an example of an inconsistent atom. No selection of atoms from $\{p_1, p_2, p_3\}$ satisfies it and, consequently, it has no models.

These examples demonstrate that abstract constraints and constraint atoms can express a broad range of constraints. In the paper, we show that abstract constraint atoms can be combined into logic program clauses to represent even more complex constraints, and that much of the theory of normal logic programs generalizes to the extended setting.

Let $\mathcal{F}$ be a class of abstract constraints over At. By an $\mathcal{F}$*-atom* we mean an abstract atom $A(X)$ such that $A \in \mathcal{F}$ and $X \subseteq At$. By an $\mathcal{F}$*-clause* we mean an expression

$$A(X) \leftarrow B_1(X_1), \ldots, B_m(X_m), \mathbf{not}(C_1(Y_1)), \ldots, \mathbf{not}(C_n(Y_n)) \qquad (1)$$

where $A(X)$, $B_i(X_i)$ and $C_j(Y_j)$ are *consistent* $\mathcal{F}$-atoms[4]. By an $\mathcal{F}$*-program* we mean a collection of $\mathcal{F}$-clauses.

[3]We recall that for an interpretation $M \subseteq At$, an atom $p \in At$ is *true* in M if $p \in M$; otherwise, p is *false* in M.

[4]The case when inconsistent atoms are allowed is not essen-

If r is a clause of the form (1), $A(X)$ is the *head* of r, denoted by $hd(r)$, and X is the *head set* of r, denoted by $hset(r)$. We also call the conjunction of literals $B_1(X_1),\ldots,B_m(X_m),\mathbf{not}(C_1(Y_1)),\ldots,\mathbf{not}(C_n(Y_n))$, the *body* of r. Finally, for an $\mathcal{F}$-program P, we define $hset(P)$ to be the union of sets $hset(r)$, for $r \in P$.

Our goals are to extend concepts, techniques and results of normal logic programming to the class of $\mathcal{F}$-programs, to provide a uniform theoretical framework for current extensions of logic programs with aggregates, and to identify basic assumptions under which such extensions are possible. To this end, we note that normal logic programs can be viewed as programs with abstract constraint atoms. Let $U_{At} = \{X \subseteq At\colon X \neq \emptyset\}$. Clearly, U_{At} is an abstract constraint and, for every atom $a \in At$ and every interpretation M, $M \models a$ if and only if $M \models U_{At}(a)$. That is, propositional atoms are *equivalent* to abstract constraint atoms $U_{At}(a)$. Under this equivalence, a normal logic program P can be viewed as a $\{U_{At}\}$-program, by regarding each atom a as a shorthand for the constraint atom $U_{At}(a)$. Formally, for a normal logic program P, we define its corresponding $\{U_{At}\}$-program P^{ac} (the superscript stands for *abstract constraints*) to consist of all $\{U_{At}\}$-clauses obtained from clauses in P by replacing atoms a with abstract constraint atoms $U_{At}(a)$. As a verification of the soundness of our approach, we will show in the paper that several properties of programs with abstract constraints, when applied to programs of the form P^{ac}, reduce to well known properties of normal logic programs.

Clauses of normal logic programs are typically regarded as computational devices: *assuming that preconditions of a clause have been established, the clause provides a justification to establish (compute) its head.* Crucial concepts behind formal accounts of that intuition are those of a Horn program, the corresponding *bottom-up* computation, and a least Herbrand model, which defines the *result* of the computation. Computations and their results are well defined due to the *monotone* behavior of Horn programs. To extend normal logic programming to the class of programs with abstract constraint atoms, one needs a generalization of the class of Horn programs supporting an appropriate notion of a computation, with the results of computations playing the same role as that played by the least Herbrand model.

In order to accomplish that, it is not enough simply to disallow the negation operator in the bodies of $\mathcal{F}$-clauses. It is also necessary to restrict the class of constraints to those that are *monotone* (that is, intuitively, once true in an interpretation, they remain true in its every superset). Without that assumption, the "monotonicity" of normal Horn programs does not generalize and there is no straightforward way to define the concept of a computation.

Formally, we say that an abstract constraint C is *monotone* if for every $A, A' \subseteq At$, if $A \in C$ and $A \subseteq A'$ then $A' \in C$ (in other words, monotone constraints are precisely upward-closed families of subsets of At). An abstract constraint atom $C(X)$ is monotone if its constraint C is monotone. The following *monotonicity* property of abstract monotone atoms is a direct consequence of these definitions.

tially different and can be reduced to the one we focus on here. We do not discuss that issue in detail due to space restrictions.

Proposition 1 *Let C be an abstract monotone constraint over At, $X \subseteq At$, and let $M, M' \subseteq At$ be two interpretations. If $M \models C(X)$ and $M \subseteq M'$, then $M' \models C(X)$.*

We note that if all the individual weights used by the weight function w are non-negative, the weight constraint $W^{\leq}_{\bar{w},v}$, which we discussed earlier, is monotone. The maximum constraint $Max^{\leq}_{\bar{w},v}$ is monotone for every weight function w. We also note that the constraint U_{At}, which we used to interpret normal logic programs as programs with abstract constraints, is monotone as well, which further justifies a central role of monotone constraints. On the other hand, we note that some common constraints, for instance, even- and odd-cardinality constraints E and O, are not monotone.

From now on, unless explicitly stated otherwise, we restrict our attention to constraints that are *monotone*. Under this assumption, the concept of a Horn program has an obvious direct generalization. Namely, for any class $\mathcal{F}$ of monotone constraints, an $\mathcal{F}$-program is a *Horn $\mathcal{F}$-program* if it contains no occurrences of the operator $\mathbf{not}$.

Nondeterministic one-step provability operator

A basic tool in the studies of normal logic programs is that of the *one-step provability operator* (van Emden & Kowalski 1976). It describes the results of updating an interpretation by computing the heads of applicable program clauses. The difficulty with extending that operator to the case of $\mathcal{F}$-programs is that once a clause "fires", there may be many sets of atoms that can be derived on that basis, as there may be many ways to "satisfy" the abstract atom in the head of the clause. In this section, we address that issue.

A *nondeterministic operator* on a set D is any function $f\colon D \to \mathcal{P}(D) \setminus \emptyset$ (that is, for every $d \in D$, $f(d) \neq \emptyset$). One can view the set $f(d)$ as the collection of all possible outcomes of applying f to d, one of which (by the definition, at least one is available) can be selected nondeterministically as the *actual* outcome of f.

Definition 1 *Let $\mathcal{F}$ be a class of monotone constraints. Let P be an $\mathcal{F}$-program and let $M \subseteq At$.*
*(1) A clause $r \in P$ is M-*applicable, *if M satisfies all literals in the body of r. We denote by $P(M)$ the set of all M-applicable clauses in P.*
(2) A set M' is nondeterministically one-step provable *from M by means of P, if $M' \subseteq hset(P(M))$ and $M' \models hd(r)$, for every clause r in $P(M)$.*
(3) The nondeterministic one-step provability operator T^{nd}_P, *is a function from $\mathcal{P}(At)$ to $\mathcal{P}(\mathcal{P}(At))$ such that for every $M \subseteq At$, $T^{nd}_P(M)$ consists of all sets M' that are nondeterministically one-step provable from M by means of P.*

Our use of the term *nondeterministic operator* in reference to T^{nd}_P is justified. We have the following property.

Proposition 2 *Let $\mathcal{F}$ be a class of monotone constraints over a set At and let P be an $\mathcal{F}$-program. For every $M \subseteq At$, $hset(P(M)) \in T^{nd}_P(M)$. In particular, $T^{nd}_P(M) \neq \emptyset$.*

We use the operator T_P^{nd} to introduce and characterize several semantics for $\mathcal{F}$-programs, and to relate the formalism of $\mathcal{F}$-programming to normal logic programming. We will often rely on the intuition that elements of $T_P^{nd}(M)$ can be viewed as possible *updates* of M on the basis of clauses in P, none of them distinguished and each available for a nondeterministic selection as an *actual* update.

We conclude this section with a characterization of models of $\mathcal{F}$-programs. It is a generalization of the familiar description of models of normal logic programs as prefixpoints of the van Emden-Kowalski operator (the conditions in the theorem are commonly used as a definition of *prefixpoints* of a nondeterministic operator).

Theorem 1 *Let $\mathcal{F}$ be a class of monotone constraints over a set At. Let P be an $\mathcal{F}$-program and let $M \subseteq At$. The set M is a model of P if and only if there is $M' \in T_P^{nd}(M)$ such that $M' \subseteq M$.*

Supported models of $\mathcal{F}$-programs

For a set M of atoms, we say that M-applicable clauses in an $\mathcal{F}$-program P provide *support* to atoms in the heads of these clauses. For M to be a model of P, M must satisfy the heads of all applicable clauses. To this end, M needs to contain some of the atoms appearing in the heads of these clauses (atoms with support in M and P) and, possibly, also some atoms that do not have such support. Models that contain only atoms with support form an important class of models generalizing the class of supported models for normal logic programs (Clark 1978; Apt 1990).

Definition 2 *Let $\mathcal{F}$ be a class of monotone constraints and let P be an $\mathcal{F}$-program. A set of atoms M is a* supported model *of P if M is a model of P and $M \subseteq hset(P(M))$.*

Supported models have the following characterization generalizing a characterization of supported models of normal logic programs as fixpoints of the the van Emden-Kowalski operator (the characterizing condition is commonly used as a definition of a *fixpoint* of a nondeterministic operator).

Theorem 2 *Let $\mathcal{F}$ be a class of monotone constraints. Let P be an $\mathcal{F}$-program. A set $M \subseteq At$ is a supported model of P if and only if $M \in T_P^{nd}(M)$.*

Our concept of a supported model generalizes that considered in normal logic programming. Specifically, supported models of a normal logic program coincide with supported models of this same program viewed as a $\{U_{At}\}$-program.

Theorem 3 *For every normal logic program P, a set of atoms M is a supported model of P if and only if M is a supported model of the $\{U_{At}\}$-program P^{ca}*

Horn $\mathcal{F}$-programs

The concepts of a model and supported model extend to $\mathcal{F}$-programs directly. To find a proper generalization of stable models is less straightforward. We will address it in the next two sections. In this section, we will study *Horn $\mathcal{F}$-programs*. The key issues is that of generalization of the concept of a bottom-up computation which, in the case of Horn $\mathcal{F}$-programs is inherently non-deterministic.

Let $\mathcal{F}$ be a class of monotone constraints and let P be a Horn $\mathcal{F}$-program. A *P-computation* is a sequence $(X_n)_{n=0,1,\ldots}$ such that $X_0 = \emptyset$ and, for every non-negative integer n:
1. $X_n \subseteq X_{n+1}$, and
2. $X_{n+1} \in T_P^{nd}(X_n)$.
Given a computation $t = (X_n)_{n=0,1,\ldots}$, we call $\bigcup_{n=0}^{\infty} X_n$ the *result* of the computation t and denote it by R_t.

The following proposition shows that the results of computations contain only atoms with support and that they are supported models (and, thus, also models) of Horn $\mathcal{F}$-programs.

Proposition 3 *Let $\mathcal{F}$ be a class of monotone constraints and let P be a Horn $\mathcal{F}$-program. For every P-computation t:*
1. $R_t \subseteq hset(P(R_t))$, and
2. R_t, is a supported model of P.

Let M be a model of P. We define $X_0^{P,M} = \emptyset$ and, for every $n \geq 0$, we set $X_{n+1}^{P,M} = hset(P(X_n^{P,M})) \cap M$.

Theorem 4 *Let $\mathcal{F}$ be a class of monotone constraints and let P be a Horn $\mathcal{F}$-program. For every model M of P, the sequence $t^{P,M}$ is a P-computation.*

We call the P-computation $t^{P,M}$, where M is a model of P, the *canonical* P-computation for M. Since every $\mathcal{F}$-program P has models, every Horn $\mathcal{F}$-program has at least one computation. In general, a Horn $\mathcal{F}$-program may have multiple computations with, possibly, different results. Indeed, when deriving X_{n+1} from what was computed so far, we can pick for X_{n+1} *any* element form the collection $T_P^{nd}(X_n)$ which, typically, contains more than one element.

We use computations to distinguish an important class of models of Horn $\mathcal{F}$-programs, which generalizes the concept of a least Herbrand model of a normal Horn program.

Definition 3 *Let $\mathcal{F}$ be a class of monotone constraints and let P be a Horn $\mathcal{F}$-program. We say that a set of atoms M is a* derivable model *of P if there exists a P-computation t such that $M = R_t$.*

It follows from Proposition 3(2) that derivable models of a Horn $\mathcal{F}$-program P are supported models (and therefore models) of P. The following theorem shows they are the results of their own canonical computations.

Theorem 5 *Let $\mathcal{F}$ be a class of monotone constraints. For every derivable model M of a Horn $\mathcal{F}$-program P, we have $M = R_{t^{P,M}}$.*

Proposition 3(2) and Theorems 4 and 5 entail additional properties of derivable models of Horn $\mathcal{F}$-programs.

Corollary 1 *Let $\mathcal{F}$ be a class of monotone constraints and let P be a Horn $\mathcal{F}$-program. Then:*
1. P has at least one derivable model
2. Every model of P contains a derivable model of P
3. Every minimal model of P is derivable.

These properties generalize their counterparts holding for normal Horn programs. Indeed, the $\{\mathcal{U}_{At}\}$-program P^{ca}, where P is a normal Horn program, has exactly one P^{ca}-computation coinciding with the bottom-up computation for P, and exactly one derivable model that coincides with the least Herbrand model of P.

Stable models of $\mathcal{F}$-programs

We will now define stable models of $\mathcal{F}$-programs. To this end, we will generalize the concept of the *reduct* (Gelfond & Lifschitz 1988) to the case of $\mathcal{F}$-programs and exploit results on Horn $\mathcal{F}$-programs from the previous section.

Definition 4 *Let $\mathcal{F}$ be a class of monotone constraints and let P be an $\mathcal{F}$-program. For a set of atoms $M \subseteq At$ we define the* reduct *of P with respect to M, P^M in symbols, as a Horn $\mathcal{F}$-program obtained from P by (1) removing from P every clause containing in the body a literal $\mathbf{not}(A)$ such that $M \models A$, and (2) removing all literals of the form $\mathbf{not}(A)$ from all remaining clauses in P.*

It is clear that the reduct P^M is a Horn $\mathcal{F}$-program. Thus, the following definition is sound, as the concept of a derivable model is well defined for the reduct.

Definition 5 *Let $\mathcal{F}$ be a class of monotone constraints and let P be an $\mathcal{F}$-program. A set of atoms M is a* stable *model of P if M is a derivable model of the reduct P^M.*

This definition does not make it explicit that a stable model of an $\mathcal{F}$-program is a model. It is however so and the use of the term *model* is indeed justified. In fact, a stronger property holds: stable models of $\mathcal{F}$-programs are supported.

Proposition 4 *Let $\mathcal{F}$ be a class of monotone constraints and P be an $\mathcal{F}$-program. If $M \subseteq At$ is a stable model of P then M is a supported model of P.*

The notion of a stable model allows us to strengthen Proposition 3(2).

Proposition 5 *Let $\mathcal{F}$ be a class of monotone constraints and P be a Horn $\mathcal{F}$-program. A set of atoms $M \subseteq At$ is a derivable model of P if and only if M is a stable model of P.*

We conclude this section by noting that this concept of a stable model of an $\mathcal{F}$-program extends the concept of a stable model of a normal logic program.

Theorem 6 *Let P be a normal logic program and let M be a set of atoms. Then, M is a stable model of P if and only if M is a stable model of the $\{U_{At}\}$-program P^{ca}.*

Deterministic $\mathcal{F}$-programs

A monotone constraint atom $C(X)$ is *deterministic* if X is a minimal element in C. Deterministic monotone constraint atoms have the following properties.

Proposition 6 *Let $C(X)$ be a deterministic constraint atom. Then $C(X)$ is consistent and, for every $M \subseteq At$, $M \models C(X)$ if and only if $X \subseteq M$.*

An $\mathcal{F}$-program P is *deterministic*, if the head of every clause in P is a deterministic constraint atom. For deterministic logic programs there is only one way to derive a set of atoms to satisfy the head of an applicable clause. Thus, computing with deterministic $\mathcal{F}$-programs does not involve nondeterminism. Indeed, we have the following result.

Proposition 7 *Let $\mathcal{F}$ be a class of monotone constraints and let P be a deterministic $\mathcal{F}$-program. Then, for every set of atoms M, $|T_P^{nd}(M)| = 1$.*

Consequently, for a deterministic $\mathcal{F}$-program P, the operator T_P^{nd} is deterministic and, so, can be regarded as an operator with both the domain and codomain $\mathcal{P}(At)$. We write T_P^d, to denote the unique operator such that for every $M \subseteq At$, $T_P^{nd}(M) = \{T_P^d(M)\}$. Models, supported models and stable models of a deterministic $\mathcal{F}$-program can be introduced in terms of the operator T_P^d in exactly the same way as the corresponding concepts are defined in normal logic programming in terms of the operator T_P. In particular, the algebraic treatment of logic programming developed in (Fitting 2002; Przymusinski 1990; Denecker, Marek, & Truszczyński 2000) applies literally to deterministic $\mathcal{F}$-programs. We note that this comment extends to 3- and 4-valued semantics of partial models, supported models and stable models (including the Kripke-Kleene semantics and the well-founded semantics)[5]. That is important as we do not have yet a convincing generalization of some of these semantics (most notably, multivalued stable-model semantics and the well-founded semantics) to the case of arbitrary $\mathcal{F}$-programs.

Complexity

In this section we will briefly discuss the complexity of deciding whether an $\mathcal{F}$-program has a supported (stable) model. For the complexity considerations we restrict to *finitary* abstract constraints, that is, constraints that consist of finite sets of atoms only.

We note that a finitary abstract constraint C gives rise to a decision problem $\mathcal{D}_C$: given a finite set $X \subseteq At$, decide whether $X \in C$. For many common constraints that decision problem belongs to the class P. It is so for all constraints considered earlier (in the case of a containment constraint, under the assumption that sets in $\mathcal{A}$ that define the constraint are given by lists of their elements).

Theorem 7 *Let $\mathcal{F}$ be a nonempty class of finitary abstract monotone constraints such that for every constraint $C \in \mathcal{F}$, the decision problem $\mathcal{D}_C$ is in* P *and at least one constraint in $\mathcal{F}$ is nonempty. Then the problem to decide whether an $\mathcal{F}$ program has a supported (stable) model is NP-complete.*

The proof of the upper bound is simple. Once we guess a candidate M for a stable model for P, we first verify that M is a model of P. Next, we compute the reduct P^M and, finally, we verify that M is a derivable model of P^M by constructing the canonical derivation for M. The last task is well defined as it is applied after we first verify that M is a model (we recall that canonical derivations are defined for models only). Moreover, all tasks can be accomplished in polynomial time (by our assumption on the class $\mathcal{F}$).

The proof of NP-completeness is harder. Due to the lack of space, we only note that to prove the claim we reduce to our problem the problem of the existence of stable models

[5]Results in (Denecker, Pelov, & Bruynooghe 2001) are related to this observation. They deal with programs (with aggregates), whose clauses have heads consisting of *single* atoms.

for normal logic programs (known to be NP-complete). The reduction is possible due to the fact that $\mathcal{F}$ contains at least one constraint C and this constraint contains at least one finite minimal model.

The arguments in the case of supported models are similar. We also note that Theorem 7 can be generalized so that to give the complexity of the problem of the existence of supported (stable) models of $\mathcal{F}$-programs in terms of the complexity of the decision problems $\mathcal{D}_C$, for $C \in \mathcal{F}$.

Discussion

In this paper, we discussed a generalization of normal logic programming, in which clauses consist of *abstract constraint atoms*, that is, expressions modeling constraints on sets. We showed that for a broad class of abstract constraints, called in the paper *monotone*, basic concepts, methods and results from normal logic programming generalize.

The results of the paper provide a formal tool to study properties of current extensions of logic programming systems with weight constraints such as those proposed in (Simons, Niemelä, & Soininen 2002). However, they apply equally well to extensions with much broader classes of constraints; the only requirement is their monotonicity. Such extended formalisms have clear modeling and computational advantages. Our work offers for them a solid theoretical basis.

The paper opens several interesting research directions. First, our complexity results indicate that the complexity of automated reasoning tasks for programs with constraints remains NP-complete as long as programs are built on the basis of constraints, for which the membership problem is in P. That indicates that an extension of algorithmic methods to compute stable models of programs with weight constraints in (Simons, Niemelä, & Soininen 2002) to programs with more general constraints might be possible. Second, due to close analogies between our approach and normal logic programming, we believe several other properties and results (splitting theorems, the concept of strong equivalence and its characterizations, Fages lemma, etc.) can be extended to the setting of programs with constraints. Finally, an important problem is to establish under what conditions the assumption of monotonicity can be dropped. Our formalism can easily be extended to the case allowing clauses with inconsistent heads and in such clauses, atoms built of *arbitrary* constraints can occur. Whether that result can be generalized in any significant fashion is an open question.

References

Aloul, F.; Ramani, A.; Markov, I.; and Sakallah, K. 2002. PBS: a backtrack-search pseudo-boolean solver and optimizer. SAT02, 346 – 353.

Apt, K. 1990. Logic programming. *Handbook of theoretical computer science*. 493–574.

Babovich, Y., and Lifschitz, V. 2002. *Cmodels package*. `http://www.cs.utexas.edu/users/tag/cmodels.html`.

Baral, C. 2003. *Knowledge representation, reasoning and declarative problem solving*.

Clark, K. 1978. Negation as failure. *Logic and data bases*, 293–322.

Dell'Armi, T.; Faber, W.; Ielpa, G.; Leone, N.; and Pfeifer, G. 2003. Aggregate functions in disjunctive logic programming: semantics, complexity, and implementation in DLV. IJCAI 2003.

Denecker, M.; Marek, V.; and Truszczyński, M. 2000. Approximations, stable operators, well-founded fixpoints and applications in nonmonotonic reasoning. *Logic-Based Artificial Intelligence*, 127–144.

Denecker, M.; Pelov, N.; and Bruynooghe, M. 2001. Ultimate well-founded and stable semantics for logic programs with aggregates. *ICLP 2001*, 212–226.

Dixon, H., and Ginsberg, M. 2002. Inference methods for a pseudo-boolean satisfiability solver. *AAAI-2002*, 635–641.

Ferraris, P., and Lifschitz, V. 2004. Weight constraints ans nested expressions. *Theory and Practice of Logic Programming, (forthcoming)*.

Fitting, M. C. 2002. Fixpoint semantics for logic programming – a survey. *Theoretical Computer Science* 278:25–51.

Gelfond, M., and Leone, N. 2002. Logic programming and knowledge representation – the A-prolog perspective. *Artificial Intelligence* 138:3–38.

Gelfond, M., and Lifschitz, V. 1988. The stable semantics for logic programs. *ICLP 1988*, 1070–1080.

Leone, N.; Pfeifer, G.; Faber, W.; Eiter, T.; Gottlob, G.; Perri, S.; and Scarcello, F. 2003. The dlv system for knowledge representation and reasoning. `http://xxx.lanl.gov/abs/cs.AI/0211004`.

Lin, F., and Zhao, Y. 2002. ASSAT: Computing answer sets of a logic program by SAT solvers. *AAAI-2002*, 112–117.

Marek, V.; Niemelä, I.; and Truszczyński, M. 2004. Characterizing stable models of logic programs with cardinality constraints. *LPNMR7*, 154–166.

Niemelä, I., and Simons, P. 1997. Smodels — an implementation of the stable model and well-founded semantics for normal logic programs. In *LPNMR4*, 420–429.

Pelov, N.; Denecker, M.; and Bruynooghe, M. 2004. Partial stable models for logic programs with aggregates. *LPNMR7*, 207–219.

Prestwich, S. 2002. Randomised backtracking for linear pseudo-boolean constraint problems. *CPAIOR-02*, 7–20.

Przymusinski, T. 1990. The well-founded semantics coincides with the three-valued stable semantics. *Fundamenta Informaticae* 13(4):445–464.

Simons, P.; Niemelä, I.; and Soininen, T. 2002. Extending and implementing the stable model semantics. *Artificial Intelligence* 138:181–234.

Turner, H. 2003. Strong equivalence made easy: Nested expressions and weight constraints. *Theory and Practice of Logic Programming* 3, (4&5):609–622.

van Emden, M., and Kowalski, R. 1976. The semantics of predicate logic as a programming language. *Journal of the ACM* 23(4):733–742.